國 立 虎 尾 科 技 大 學
機 械 設 計 工 程 系

專 題 製 作 報 告

強化學習在機電系統設計與控制中之應用

Application of reinforcement learning in design and control of mechatronic systems

指導教授： 嚴 家 銘 老 師
班 級： 四 設 三 甲
學 生： 李 正 揚 (40723110)
林 于 哲 (40723115)
黃 奕 慶 (40723138)
鄭 博 鴻 (40723148)
簡 國 龍 (40723150)

中華民國 １１０ 年 ３ 月

摘要

產業中需要加速許多工法的演算，以達到最佳化，但不能以實體一直測試不同方法，成本與時間不允許，便可以利用許多感測器觀測數值，以類神經網路運算，在虛擬環境架設結構，遠端控制、更改數值。

此專題是利用現成裝置冰球台，設置對應虛擬模擬環境，減少現實模擬參數設置、成本，再加入類神經網路之中的 Policy gradient 與 Reinforcement Learning，訓練冰球達到對應最佳化。

關鍵字:Policy gradient、虛擬環境架設結構、Reinforcement Learning

誌謝

致謝內容

目　錄

圖 表 目 錄

第一章 機器學習的訓練方法

1.1 類神經網絡

類神經網路的概念來源是觀察人類的中樞神經系統而啟發的，類神經網路主要由多個節點組成，每個節點稱為神經元（neurons），神經元與神經元連結而形成的網路稱為神經網路。神經元主要又分成三層：輸入層（input layer）、隱藏層（hidden layer）和輸出層（output layer）。神經元會接收上一層的訊號，並經過啟動函數（activation function）計算並輸出至下一層神經元。神經網路輸入層的神經元負責接收訊號，透過隱藏層來運算，再傳遞到輸出層執行動作。由於每個神經元經過啟動函數的非線性計算，所以神經元傳出的數值也是非線性。神經元與神經元間連線在生物學稱為突觸（Synapse），在數學模型裡每個突觸都有權重（weights）。在既定框架下，調配權重值和啟動函數會使結果更接近預期。神經網路的架構指標為以下幾項：階層數、每層神經元個數、神經元連接方式、啟動函數的類型等。

神經訊號傳遞分為前饋（feed-forward）和回饋（backpropagation）。

前饋為傳遞經過啟動函數計算的數值。回饋則是修正權重和偏差（biases），以提高下次前饋計算的準確度。

1.1.1 啟動函數

- Sigmoid Function
  An activation function.It is a key part of Nerual Network and it can be differentiable.It can make the Nenual Network unliner.

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- SigmoidPrime Function
  It is an differential Sigmoid Funtion. It can reduce error of gradient so it is some kind of loss function and Let's take a look the proof of SigmoidPrime

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma^{'}(x) = \frac{d}{dx}\sigma(x) = \frac{d}{dx}\frac{1}{1+e^{-x}} = \frac{d}{dx}(1+e^{-x})^{-1}$$

  Tip: find $f^{'}(x)$ if $f(x) = \frac{A}{B+Ce^{x}}$

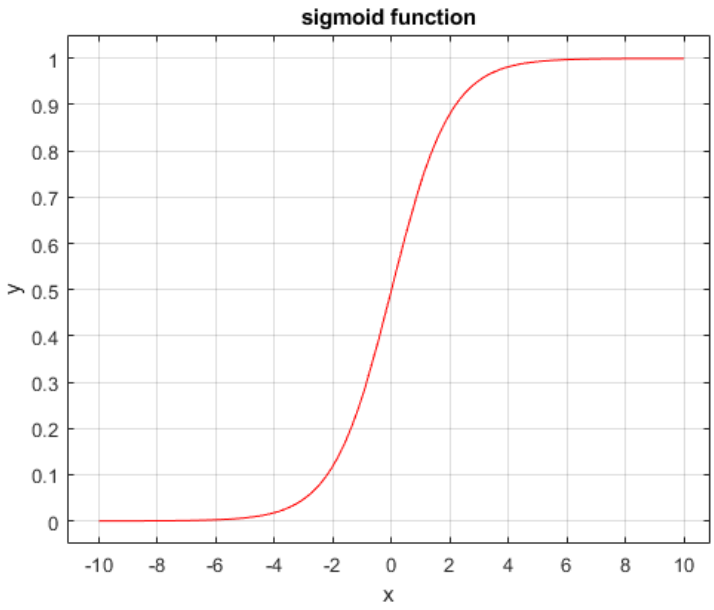


Figure 1.1: Sigmoid; from: Toward Data Science

Answer:

$$\frac{d}{dx}[\frac{1}{g(x)}] = \frac{1^{'}g(x) - 1g^{'}(x)}{g(x)^2} = \frac{g^{'}(x)}{[g(x)]^2}$$

if $g(x)$=constant

$$\frac{d}{dx}[\frac{g(x)}{h(x)}] = \frac{g^{'}(x)h(x) - g(x)h^{'}(x)}{h(x)^2} = \frac{-kh^{'}(x)}{[h(x)]^2}$$

$$f^{'}(x) = \frac{-A[\frac{d}{dx}(B + Ce^x)]}{(B + Ce^x)^2} = \frac{-A(0 + Ce^x)}{(B + Ce^x)^2} = \frac{-ACe^x}{(B + Ce^x)^2}$$

$$- - - - - - - - - - - - - - - skip - - - - - - - - - - - - - -$$

Hence:

$$= -(1 + e^{-x})^{-2}\frac{d}{dx}(1 + e^{-x}) = -(1 + e^{-x})^{-2}[\frac{d}{dx}(1) + \frac{d}{dx}(e^{-x})]$$

$$= -(1+e^{-x})^{-2}[0+\frac{d}{dx}(e^{-x})] = -(1+e^{-x})^{-2}[\frac{d}{dx}(e^{-x})] = -(1+e^{-x})^{-2}[e^{-x}\frac{d}{dx}(-x)]$$

$$-(1+e^{-x})^{-2}[e^{-x}(-1)] = -(1+e^{-x})^{-2}(-e^{-x}) = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1(e^{-x})}{(1 + e^{-x})(1 + e^{-x})}$$

$$= \frac{1}{1 + e^{-x}}\frac{e^{-x}}{1 + e^{-x}} = \frac{1}{1 + e^{-x}}\frac{e^{-x} + 1 - 1}{1 + e^{-x}} = \frac{1}{1 + e^{-x}}(\frac{1 + e^{-x}}{1 + e^{-x}} - \frac{1}{1 + e^{-x}})$$

$$= \sigma(x)[1 - \sigma(x)]$$

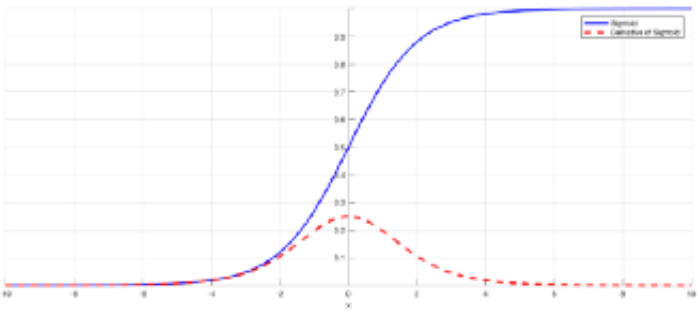


Figure 1.2: SigmoidPrime; from: Toward Data Science

- Relu Function
  This is the most commly used activation function in nerual network.It also can solve the Gradient Descent problem.

$$f(x) = max(0, x)$$
$$if, x < 0, f(x) = 0$$
$$elsef(x) = x$$

- Adam Function
  Combine the advantage of Adagrad and RMSprop.The paper contained some very promising diagrams, showing huge performance gains in terms of speed of training but in some cases Adam actually finds worse solution than stochastic gradient descent. Let's take a look caculation prosess

$$g_t = \delta_\theta f(\theta)$$

  First moment exponentially moving averages : $m_t = \beta(m_{t-1}) + (1 - \beta_1)(\nabla w_t)$

  $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$

  Second moment exponentially moving averages :$v_t = \beta_2(v_t - 1) + (1 - \beta_2)(\nabla w_t)^2$

  $\hat{v_t} = \frac{v_t}{1-\beta_2^t}$

  Hence,Adam Function:

$$\omega_{t-1} = \omega_t - \frac{\eta}{\sqrt{\hat{v}_t - \epsilon}}\hat{m}_t$$

- Mean Squared Error
  It tells you how close a regression lines to a set of points.It does this by taking the distances from the points to the regression line (these distances are the "errors") and squaring them.The squaring is necessary to remove any negative signs. It also gives more weight to larger differences.

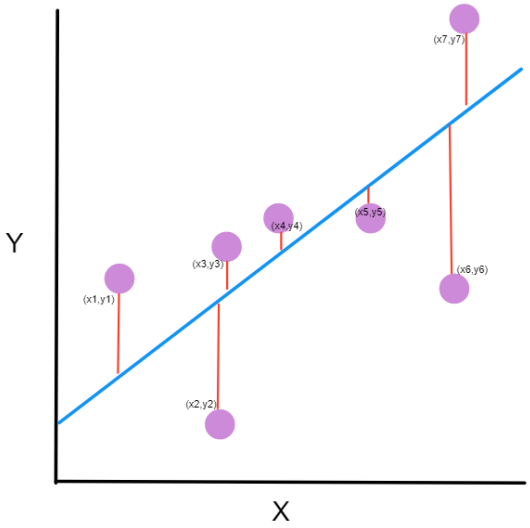


Figure 1.3: regression line

(Extracted from here).

regression line :It is a line of the minimize distance of data points.
n : number of data points
y_i : observed values
$\hat{y}_i$: predicted values

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y}_i)^2$$

1.2 深度學習

- Deep learning 會在此時崛起?
  Four major trends in scientific computing have become increasingly important for deep learning.

  (1) 支援張量運算程式語言與程式庫的崛起
  First, starting in the 1960s, the development of domain specific languages such as APL, MATLAB, R and Julia, turned multidimensional arrays (often referred to as tensors) into first-class objects supported by a comprehensive set of mathematical primitives (or operators) to manipulate them.
  Separately, libraries such as NumPy, Torch, Eigen and Lush made array-based programming productive in general purpose languages such as Python, Lisp, C++ and Lua.

  (2) 自動求導數套件的開發
  Second, the development of automatic differentiation made it possible to fully automate the daunting labor of computing derivatives. This made it significantly easier to experiment with different machine learning approaches while still allowing for efficient gradient based optimization. The autograd package popularized the use of this technique for NumPy arrays, and similar approaches are used in frameworks such as Chainer, DyNet, Lush, Torch, Jax and Flux.jl.

  (3) 自由開源軟體的普及
  Third, with the advent of the free software movement, the scientific community moved away from closed proprietary software such as Matlab, and towards the open-source Python ecosystem with packages like NumPy, SciPy, and Pandas. This fulfilled most of the numerical analysis needs of researchers while allowing them to take advantage of a vast repository of librariesto handle dataset preprocessing, statistical analysis, plotting, and more.
  Moreover, the openness, interoperability, and flexibility of free software fostered the development of vibrant communities that could quickly address new or changing needs by extending the existing functionality of a library or if needed by developing and releasing brand new ones. While there is a rich offering of open-source software for neural networks in languages other than Python, starting with Lush in Lisp, Torch in C++, Objective-C and Lua, EBLearn in C++, Caffe in C++, the network effects of a large ecosystem such as Python made it an essential skill to jumpstart one’s research. Hence, since 2014,most deep learning frameworks converged on a Python interface as an essential feature.

  (4) 多核 GPU 運算的發展
  Finally, the availability and commoditization of general-purpose massively parallel hardware such as GPUs provided the computing power required by deep learning methods. Specialized librariessuch as cuDNN, along with a body of academic work (such as Andrew Lavin. maxdnn: An efficient convolution kernel for deep learning with maxwell gpus,January 2015 and Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks.2016 IEEEConference on Computer Vision and Pattern Recognition (CVPR), pages 4013–4021, 2016), produced aset of high-performance reusable deep learning kernels that enabled frameworks such as Caffe,Torch7, or TensorFlow to take advantage of these hardware accelerators.

1.3 強化學習

- What is Reinforcement Learning?
  強化學習是通過 agent 與已知或未知的環境持互動，不斷適應與學習，得到的回饋可能是正面，也就是 reward，如果得到負面，那就是 punishments。考慮到 agent 與環境互動，我們就能決定要執行哪個動作。簡而言之，強化學習是建立在 reward 與 punishments 上。

  The key point of Reinforcement Learning:

  - It differs from normal Machine Learning, as we do not look at training datasets.
  - Interaction happens not with data but with environments, through which we depict real-world scenarios
  - As Reinforcement Learning is based on environments, many parameters come in to play. It takes lots of information to learn and act accordingly.
  - Environments in Reinforcement Learning are real-world scenarios that might be 2D or 3D simulated worlds or gamebased scenarios.
  - Reinforcement Learning is broader in a sense because the environments can be large in scale and there might be a lot of factors associated with them.
  - The objective of Reinforcement Learning is to reach a goal.
  - Rewards in Reinforcement Learning are obtained from the environment.

- Faces of Reinforcement Learning
- The Flow of Reinforcement Learning
  The key points of consideration:

  - The Reinforcement Learning cycle works in an interconnected.
  - The distinct communication happens with rewards in mind.
  - There is distinct communication between the agent and the environment.
  - The object or robot moves from one state to another.
  - An action is taken to move from one state to another

  The agent is also a decision maker because it tries to take an action that will get it the maximum reward.
  When the agent starts interacting with the environment, it can choose an action and respond accordingly. From then on, new scenes are created. When the agent changes from one place to another in an environment, every change results in some kind of modification. These changes are depicted as scenes. The transition that happens in each step helps the agent solve the Reinforcement Learning problem more effectively

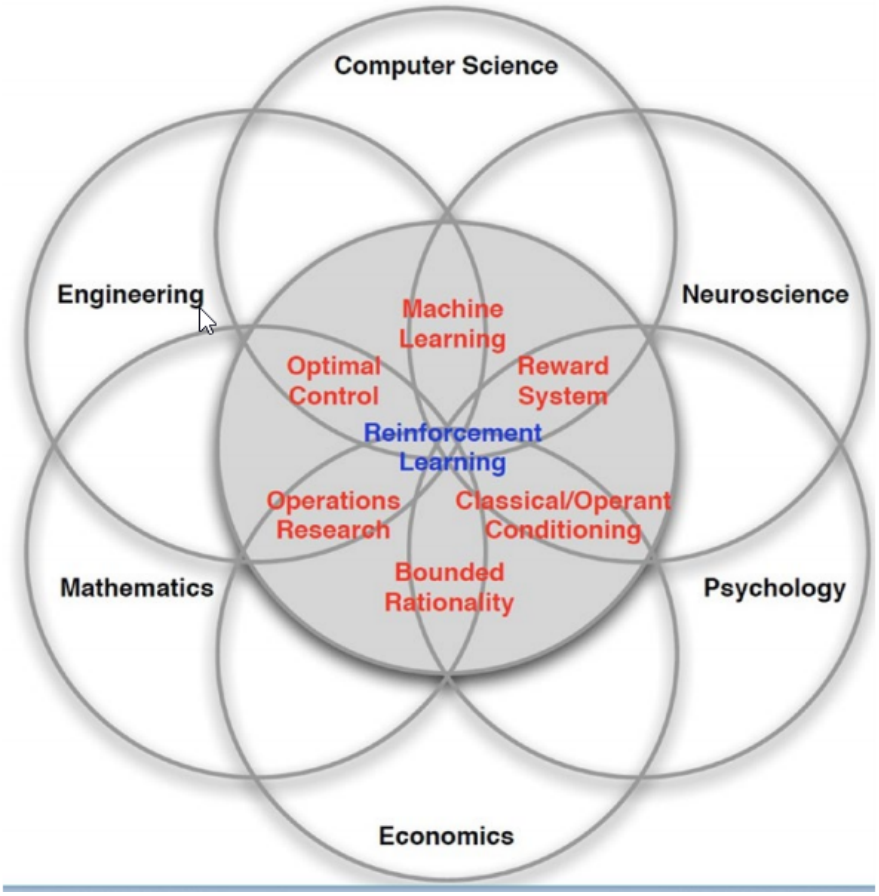


Figure 1.4: Venn diagram;

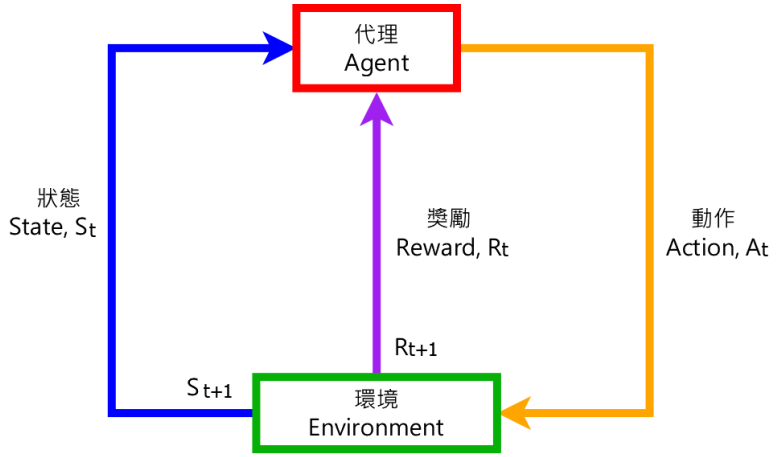


Figure 1.5: RL structur

- Different Terms in Reinforcement Learning
  There are two constants that are important in this case—gamma (γ) and lambda (λ)

  Gamma is used in each state transition and is a constant value at each state change. Gamma allows you to give information about the type of reward you will be getting in every state. Gamma (γ) is called a discount factor and it determines what future reward types we get:

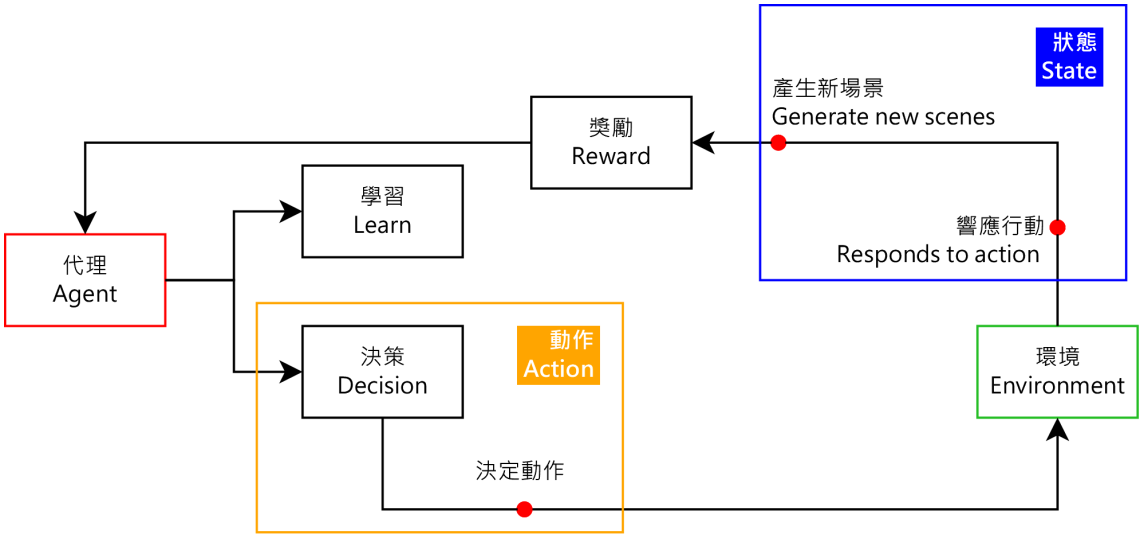


Figure 1.6: The entire interaction process;

- A gamma value of 0 means the reward is associated with the current state only.
- A gamma value of 1 means that the reward is long-term.

Lambda is generally used when we are dealing with temporal difference problems. It is more involved with predictions in successive states. Increasing values of lambda in each state shows that our algorithm is learning fast. The faster algorithm yields better results when using Reinforcement Learning techniques. As you' ll learn later, temporal differences can be generalized to what we call TD(Lambda).

- Interactions with Reinforcement Learning

  The interactions between the agent and the environment occur with a reward. We need to take an action to move from one state to another.
  Reinforcement Learning is a way of implementing how to map situations to actions so as to maximize and find a way to get the highest rewards. The machine or robot is not told which actions to take, as with other forms of Machine Learning, but instead the machine must discover which actions yield the maximum reward by trying them.

- How Reward Works
  A reward is some motivator we receive when we transition from one state to another. It can be points, as in a video game. The more we train, the more accurate we become, and the greater our reward.

- Agents
  In terms of Reinforcement Learning, agents are the software programs that make intelligent decisions. Agents should be able to perceive what is happening in the environment. Here are the basic steps of the agents:
  - When the agent can perceive the environment, it can make better decisions.
  - The decision the agents take results in an action.

– The action that the agents perform must be the best, the optimal, one.

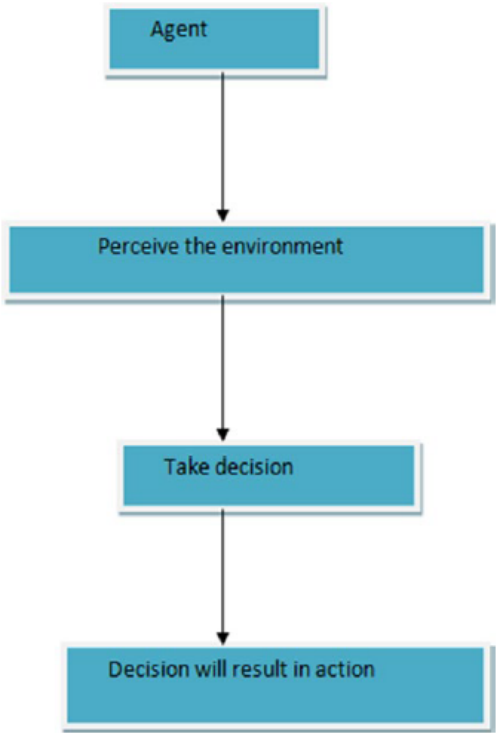


Figure 1.7: agent

- RL Environments
  The environments in the Reinforcement Learning space are comprised of certain factors that determine the impact on the Reinforcement Learning agent. The agent must adapt accordingly to the environment. These environments can be 2D worlds or grids or even a 3D world.
  Here are some important features of environments:
  - Deterministic
  - Observable
  - Discrete or continuous
  - Single or multiagent.

1.4 深度強化學習

1.4.1 Deep Reinforcement Learning

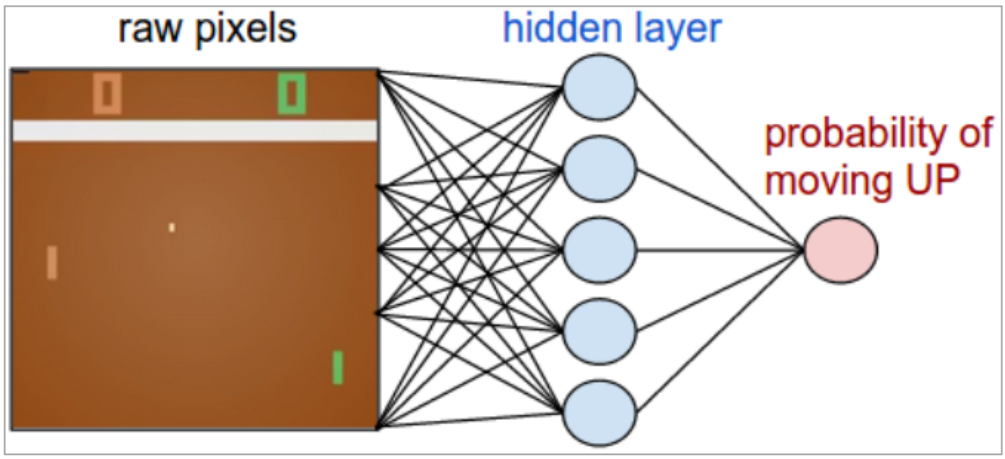


Figure 1.8: Our network is a 2-layer fully-connected net.

we're going to define a policy network that implements our player (or "agent"). This network will take the state of the game and decide what we should do (move UP or DOWN). As our favorite simple block of compute we'll use a 2-layer neural network that takes the raw image pixels (100,800 numbers total (210*160*3)), and produces a single number indicating the probability of going UP. Note that it is standard to use a stochastic policy, meaning that we only produce a probability of moving UP. Every iteration we will sample from this distribution (i.e. toss a biased coin) to get the actual move. The reason for this will become more clear once we talk about training.

1.4.2 Supervised Learning

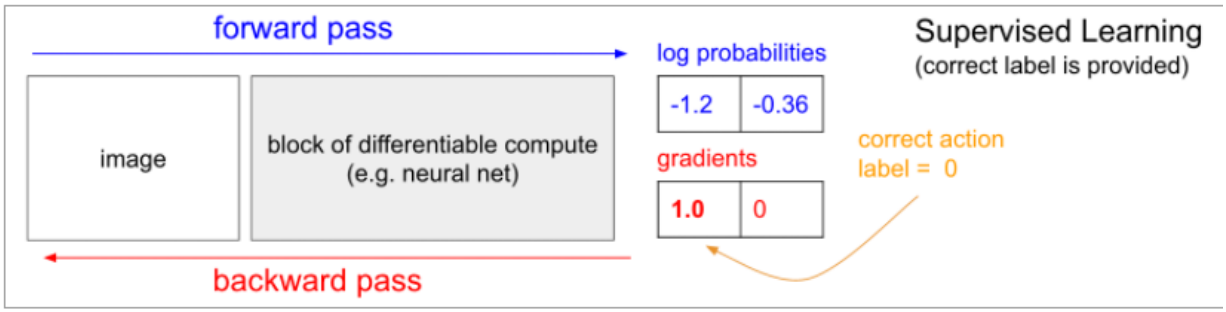


Figure 1.9: supervising learning

Before we dive into the Policy Gradients solution I'd like to remind you briefly about supervised learning because, as we'll see, RL is very similar. Refer to the diagram below. In ordinary supervised learning we would feed an image to the network and get some probabilities, e.g. for two classes UP and DOWN. I'm showing log probabilities (-1.2, -0.36) for UP and DOWN instead of the raw probabilities (30 persent and 70 persent in this case) because we always optimize the log probability of the correct label (this makes math nicer,

and is equivalent to optimizing the raw probability because log is monotonic). Now, in supervised learning we would have access to a label. For example, we might be told that the correct thing to do right now is to go UP (label 0). In an implementation we would enter gradient of 1.0 on the log probability of UP and run backprop to compute the gradient vector $Wlogp(y = UP|x)$. This gradient would tell us how we should change every one of our million parameters to make the network slightly more likely to predict UP. For example, one of the million parameters in the network might have a gradient of -2.1, which means that if we were to increase that parameter by a small positive amount (e.g. 0.001), the log probability of UP would decrease by 2.1 * 0.001 (decrease due to the negative sign). If we then did a parameter update then, our network would now be slightly more likely to predict UP when it sees a very similar image in the future.

1.4.3 Log Derivative Trick

機器學習涉及操縱機率。這個機率通常包含 normalised-probabilities 或 log-probabilities。能加強解決現代機器學習問題的關鍵點，是能夠巧妙的在這兩種型式間交替使用，而對數導數技巧就能夠幫助我們做到這點，也就是運用對數導數的性質。

1.4.4 Score Functions

對數導數技巧的應用規則是基於參數 θ 梯度的對數函數 $p(x:\theta)$，如下:

$$\nabla_\theta logp(x:\theta) = \frac{\nabla_\theta p(x:\theta)}{p(x:\theta)}$$

$p(x:\theta)$ 是 likelihood ; function 參數 θ 的函數，它提供隨機變量 x 的概率。在此特例中，$\nabla_\theta logp(x:\theta)$ 被稱為 Score Function，而上述方程式右邊為 score ratio(得分比)。

The score function has a number of useful properties:

- The central computation for maximum likelihood estimation. Maximum likelihood is one of the dominant learning principles used in machine learning, used in generalised linear regression, deep learning, kernel machines, dimensionality reduction, and tensor decompositions, amongst many others, and the score appears in all these problems.
- The expected value of the score is zero. Our first use of the log-derivative trick will be to show this.

$$\mathbb{E}_{p(x;\theta)}[\nabla_\theta \log p(\mathbf{x};\theta)] = \mathbb{E}_{p(x;\theta)}\left[\frac{\nabla_\theta p(\mathbf{x};\theta)}{p(\mathbf{x};\theta)}\right]$$

$$= \int p(\mathbf{x};\theta)\frac{\nabla_\theta p(\mathbf{x};\theta)}{p(\mathbf{x};\theta)}dx = \nabla_\theta \int p(\mathbf{x};\theta)dx = \nabla_\theta 1 = 0$$

In the first line we applied the log derivative trick and in the second line we exchanged the order of differentiation and integration. This identity is the type of probabilistic flexibility we seek: it allows us to subtract any term from the score that has zero expectation, and this modification will leave the expected score unaffected (see control variates later).

- The variance of the score is the Fisher information and is used to determine the Cramer-Rao lower bound.

$$\mathbb{V}[\nabla_\theta \log p(\mathbf{x};\theta)] = \mathcal{I}(\theta) = \mathbb{E}_{p(x;\theta)}[\nabla_\theta \log p(\mathbf{x};\theta) \nabla_\theta \log p(\mathbf{x};\theta)^\top]$$

  We can now leap in a single bound from gradients of a log-probability to gradients of a probability, and back. But the villain of today's post is the troublesome expectation-gradient of Trick 4, re-emerged. We can use our new-found power—the score function—to develop yet another clever estimator for this class of problems.

1.4.5 Score Function Estimators

Our problem is to compute the gradient of an expectation of a function f:

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \nabla_\theta \int p(z;\theta) f(z) dz$$

This is a recurring task in machine learning, needed for posterior computation in variational inference, value function and policy learning in reinforcement learning, derivative pricing in computational finance, and inventory control in operations research, amongst many others.

This gradient is difficult to compute because the integral is typically unknown and the parameters θ, with respect to which we are computing the gradient, are of the distribution p(z;θ). Furthermore, we might want to compute this gradient when the function f is not differentiable. Using the log derivative trick and the properties of the score function, we can compute this gradient in a more amenable way:

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \mathbb{E}_{p(z;\theta)}[f(z) \nabla_\theta \log p(z;\theta)]$$

This is a recurring task in machine learning, needed for posterior computation in variational inference, value function and policy learning in reinforcement learning, derivative pricing in computational finance, and inventory control in operations research, amongst many others.

This gradient is difficult to compute because the integral is typically unknown and the parameters θ, with respect to which we are computing the gradient, are of the distribution p(z;θ). Furthermore, we might want to compute this gradient when the function f is not differentiable. Using the log derivative trick and the properties of the score function, we can compute this gradient in a more amenable way:

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \mathbb{E}_{p(z;\theta)}[f(z) \nabla_\theta \log p(z;\theta)]$$

Let's derive this expression and explore the implications of it for our optimisation problem.

To do this, we are will use one other ubiquitous trick, a probabilistic identity trick, where we multiply our expressions by 1—a one formed by the division of a probability density by itself. Combining the identity trick with the log-derivative trick, we obtain a score function estimator for the gradient:

$$\nabla_\theta \mathbb{E}_{p(z;\theta)}[f(z)] = \int \nabla_\theta p(z;\theta) f(z) dz$$

$$= \int \frac{p(z;\theta)}{p(z;\theta)} \nabla_\theta p(z;\theta) f(z) dz$$

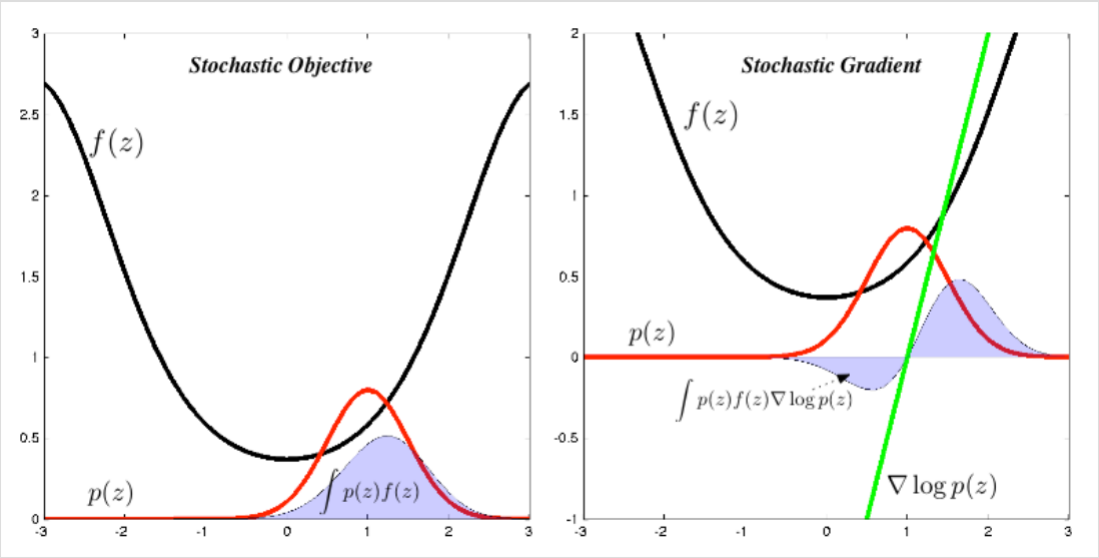


Figure 1.10: gradient change

$$= \int p(z;\theta)\nabla_\theta \log p(z;\theta) f(z) dz = \mathbb{E}_{p(z;\theta)}[f(z)\nabla_\theta \log p(z;\theta)]$$

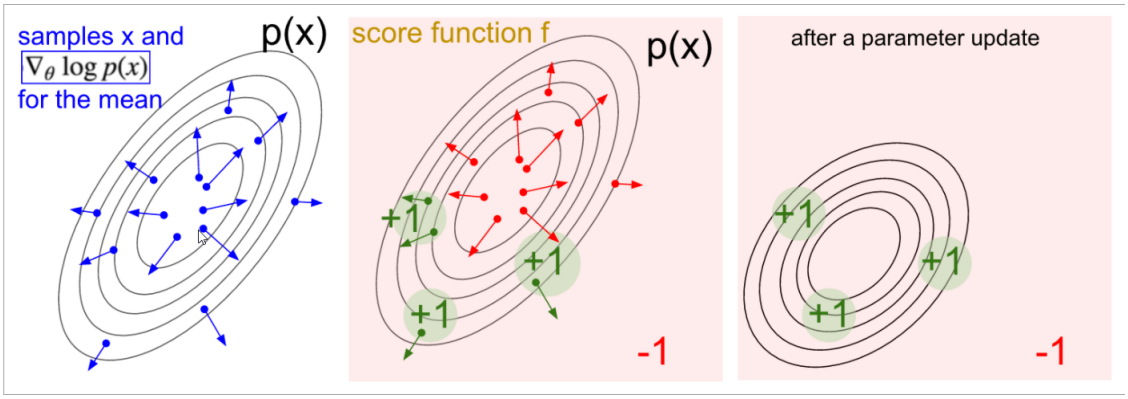
$$= \int p(z;\theta)\nabla_\theta \log p(z;\theta) f(z) dz = \mathbb{E}_{p(z;\theta)}[f(z)\nabla_\theta \log p(z;\theta)]$$

$$\approx \frac{1}{S}\sum_{s=1}^{S} f(z^{(s)})\nabla_\theta \log p(z^{(s)};\theta) \quad z^{(s)} \sim p(z)$$

A lot has happened in these four lines. In the first line we exchanged the derivative and the integral. In the second line, we applied our probabilistic identity trick, which allowed us to form the score ratio. Using the log-derivative trick, we then replaced this ratio with the gradient of the log-probability in the third line. This gives us our desired stochastic estimator in the fourth line, which we computed by Monte Carlo by first drawing samples from p(z) and then computing the weighted gradient term.

To put this in English, we have some distribution $p(x;\theta)$ (I used shorthand $p(x)$ to reduce clutter) that we can sample from (e.g. this could be a gaussian). For each sample we can also evaluate the score function f which takes the sample and gives us some scalar-valued score. This equation is telling us how we should shift the distribution (through its parameters θ) if we wanted its samples to achieve higher scores, as judged by f. In particular, it says that look: draw some samples x, evaluate their scores f(x), and for each x also evaluate the second term $\nabla_\theta logp(x;)$. What is this second term? It' s a vector - the gradient that' s giving us the direction in the parameter space that would lead to increase of the probability assigned to an x. In other words if we were to nudge θ in the direction of $\nabla_\theta logp(x;)$ we would see the new probability assigned to some x slightly increase. If you look back at the formula, it' s telling us that we should take this direction and multiply onto it the scalar-valued score $f(x)$. This will

make it so that samples that have a higher score will "tug" on the probability density stronger than the samples that have lower score, so if we were to do an update based on several samples from p the probability density would shift around in the direction of higher scores, making highly-scoring samples more likely. A visualization of the score function gradient estimator. Left: A



gaussian distribution and a few samples from it (blue dots). On each blue dot we also plot the gradient of the log probability with respect to the gaussian's mean parameter. The arrow indicates the direction in which the mean of the distribution should be nudged to increase the probability of that sample. Middle: Overlay of some score function giving -1 everywhere except +1 in some small regions (note this can be an arbitrary and not necessarily differentiable scalar-valued function). The arrows are now color coded because due to the multiplication in the update we are going to average up all the green arrows, and the negative of the red arrows. Right: after parameter update, the green arrows and the reversed red arrows nudge us to left and towards the bottom. Samples from this distribution will now have a higher expected score, as desired.

1.5 比較個方法的特色

第二章 優化器算法

2.1 Markov Chain

當前決策只會影響下個狀態，當前狀態轉移 (action) 到其他狀態的機率有所差異。

2.2 Markov Reward Process

action 到指定狀態會獲得獎勵。

$$R(s_t = s) = \mathbb{E}[r_t | s_t = s]$$

$$\gamma \in [0, 1]$$

- Horizon：
  在無限的狀態以有限的狀態表示
- Return：
  越早做出正確決策獎勵越高
  $$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... + \gamma^{T-t-1} R_T$$
- State value function：
  決策價值
  $$V_t(S) = \mathbb{E}[G_t | s_t = s]$$

$$P(s_{s+1} = s' | s_t = s, a_t = a)$$

2.3 Markov Decision Process

在 MRP 中加入決策 (decision) 和動作 (action)

- S：state 狀態
- A：action 動作
- P：狀態轉換

$$P(s_{s+1} = s' | s_t = s, a_t = a)$$

第三章 訓練環境

3.1 openAI Gym

Gym 是用於開發和比較強化學習算法的工具包，他不對 agent 的結構做任何假設，並且與任何數據計算庫兼容，而可以用來制定強化學習的算法。這個環境具有共享的介面，使我們能用來編寫常規算法，也就能教導 agents 如何步行到玩遊戲。

3.2 Pong

取自 1977 年發行的一款家用遊戲機 ATARI 2600 中的遊戲，內建於 Gym，這是一個橫向的乒乓遊戲，左方是遊玩者，右邊是馬可夫決策的特例，每個邊緣都會給予 reward(figue1)，目標就是計算再任意階段動作最佳路徑，已獲得 rewardd 最大值。

3.3 Abstract

I'd like to walk you through Policy Gradients (PG), our favorite default choice for attacking RL problems at the moment. If you’re from outside of RL you might be curious why I’m not presenting DQN instead, which is an alternative and better-known RL algorithm, widely popularized by the ATARI game playing paper. It turns out that Q-Learning is not a great algorithm (you could say that DQN is so 2013 . In fact most people prefer to use Policy Gradients, including the authors of the original DQN paper who have shown Policy Gradients to work better than Q Learning when tuned well. PG is preferred because it is end-to-end: there's an explicit policy and a principled approach that directly optimizes the expected reward. Anyway, as a running example we'll learn to play an ATARI game (Pong!) with PG, from scratch, from pixels, with a deep neural network, and the whole thing is 130 lines of Python only using numpy as a dependency (Gist link). Lets get to it.

3.4 Pong from pixels

Left: The game of Pong. Right: Pong is a special case of a Markov Decision Process (MDP): A graph where each node is a particular game state and each edge is a possible (in general probabilistic) transition. Each edge also gives a reward, and the goal is to compute the optimal way of acting in any state to maximize rewards.

The game of Pong is an excellent example of a simple RL task. In the ATARI 2600 version we’ll use you play as one of the paddles (the other is controlled by a decent AI) and you have to bounce the ball past the other player (I don’t really have to explain Pong, right?). On the low level the game works as follows: we

receive an image frame (a 210x160x3 byte array (integers from 0 to 255 giving pixel values)) and we get to decide if we want to move the paddle UP or DOWN (i.e. a binary choice). After every single choice the game simulator executes the action and gives us a reward: Either a +1 reward if the ball went past the opponent, a -1 reward if we missed the ball, or 0 otherwise. And of course, our goal is to move the paddle so that we get lots of reward.

第四章　模擬環境

第五章 伺服器

- Oracle VM VirtualBox
  因應在虛實整合需要用到虛擬環境去模擬的情況，我們使用 Ubuntu 當作我們的作業系統去建構虛擬環境，而在此之前要先安裝「虛擬機器工作站」—Oracle VM VirtualBox 。

5.1 Oracle VM VirtualBox 介紹

當要使用不同的的作業系統 (Linux , Ubuntu , Red Hat ...) 而不想與其資料存放時共用一個硬碟 (沒有多餘硬碟, 不想硬碟之間有資料重疊...) 時，就可以使用其軟體做練習，降低操作失誤帶來的成本，而此軟體目前為免費，並隨時會更新，另外其特色有:

- 只要自備作業系統 (光碟片, ISO 映像檔) ，即可在啟動 Oracle VM VirtualBox 後直接開啟要操作的執行檔 (作業系統)，不必再把主機本身重新關機，當然開啟多個作業系統之間也有共通性，可直接從視窗 A 做網路、檔案分享、複製貼上等動作到視窗 B。
- 除了作業系統裡面的執行，還可在其中練習磁碟分割、格式化以及 BIOS 啟動等 (但是未支援 USB 啟動) 。
- 空間的佔用上並不是真實佔用空間，而是依據使用者的操作而變化 (使用者用多少就是多少)。相對的，使用者雖然一開始設定該虛擬電腦的記憶體大小與硬碟空間是實時依據操作者決定，但終究還是佔掉電腦效能，所以 VirtualBox 的效能還是依據電腦本身的硬體配備
-

第六章　深度強化學習的訓練與控制

第七章　問題與討論

參考文獻

[1] https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e
[2] https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

作者簡介

test